REMARKS

Pending Claims

Claims 1, 5, 6, and 8-24 are still pending, of which claims 1, 15, and 16 are independent.

Claim Rejections Under 35 U.S.C. 103(a)

The Examiner has rejected all of the pending claims as being unpatentable under 35 U.S.C. 103(a) over Yates, et al. (US 6,549,959 – "*Yates*") in view of Robinson et al. (US 5,432,795 – "*Robinson*").

With respect to each of the independent claims 1, 15, and 16, the Examiner wrote:

> Yates fails to disclose specifically delaying handling of each asynchronous sensed exception. However, Robinson discloses delaying handling of each asynchronous sensed exception (col. 9, lines 15-25). It would have been obvious to one of ordinary skill in the art at the time of invention was made to improve upon exception handler taught by Yates' provides a flexibility to have a single common strategy for processing all exceptions raised during the binary translation either by source or target ISA and execute exception handler on either source ISA or target ISA.

The cited portion of *Robinson* (col. 9, lines 15-25) reads as follows:

> 2. If an asynchronous event occurs after Y instructions in the groups G1 and G2 have been executed and if there are no Y instructions in the group G3, or if the event occurs after execution of all Y instructions included in the group G3 as indicated by an arrow 73 in FIG. 6, the processing of the asynchronous event can be briefly delayed as the group G4 instructions are executed with foreknowledge that no state exceptions are possible. Again execution is provided with X instruction granularity.

This portion of *Robinson's* disclosure illustrates two special limitations of *Robinson's* system:

First, *Robinson* presupposes that translated code ("Y code") is arranged according to a specific "ordering criteria":

> The translation results are generated in accordance with the present invention as Y code 45 which is ordered to preserve hard guarantees of the code being translated and particularly to facilitate a guaranteed preservation of X instruction granularity when the Y code is actually executed. (Col. 7, lines 62-67)

and

> Next, as indicated by functional block 68, the resultant Y code is ordered in accordance with predetermined criteria that facilitate preservation of X instruction granularity during subsequent actual Y code execution. (Col. 8, lines 23-27)

The four groups G1-G4 into which *Robinson* arranges Y code are defined in col. 8, lines 33-53:

> When an X instruction is translated to "many" Y instructions, the ordering criteria 52 (FIG. 2) employed to organize the Y instructions preferably are those that group and order the Y instructions in the Y code for the currently translated X instruction (granule) as follows:
>
> 1. A first group G1 of instructions in the Y code are those that get inputs and place those inputs in temporary storage.
> 2. A second group G2 of instructions in the Y code are those that operate on inputs, generate modified results, and store those results to temporary storage.
> 3. A third group G3 of instructions in the Y code are those that update X state (memory or register) and are subject to possible exceptions (as defined hereinafter).
> 4. A fourth and last group G4 of instructions in the Y code are those that update X state (memory or register) and are free of possible exceptions.

Second, how *Robinson* handles an asynchronous exception depends on at which point in the Y code it is sensed. The *Robinson* text cited by the Examiner mentions what *Robinson* does in one instance: ***if*** group G1 and group G2 instructions have executed, and ***if*** there are no group G3 instructions (or ***if*** all the G3 instructions have also been executed), then and only then does *Robinson* teach "briefly" delaying the processing of the asynchronous event until the group G4 instructions are executed. This is because *Robinson's* system then knows that, in this one case, "no state exceptions are possible."

Consider, however, what *Robinson* does if the asynchronous event occurs during execution of group G1 or group G2 instructions (col. 8, line 65, to col. 9, line 5, emphasis added):

> 1. If an asynchronous event occurs during Y code execution at any time during the execution of the first two groups G1 and G2 of Y instructions as shown by an arrow 71 in the diagram of FIG. 6, X instruction granularity is maintained by permitting asynchronous event processing and ***backing up the Y instruction counter*** PC to the next ***backup*** Y instruction (Y_0 in FIG. 6) that is a granular boundary for an X instruction.

Robinson explains why he backs up – that is, ***rolls back*** – execution of the current translated target (Y) instruction sequence in the immediately following text (col. 9, lines 6-15, emphasis added):

> By ***aborting execution of the current Y code sequence for a retry***, possible breakage of X instruction granularity is avoided because a possible state access failure is avoided in any group G3 instruction in the current Y code sequence. However, as a result of the Y code organization, only temporary storage locations are erased and X instruction granularity is preserved since execution of the current Y code granule has been delayed until it can be processed with instruction granularity after the asynchronous event processing.

This restriction is also described in col. 12, lines 40-52 (emphasis added):

> If in testing successive Y instructions in the forward scan, and, if all scanned Y instructions show no exceptions (block 116), the remaining Y instructions are executed before asynchronous event processing is enabled by block 108 (FIG. 5A) without breaking X instruction granularity as previously described. On the other hand, ***if a forwardly scanned Y instruction shows an exception under the test by the block 118, functional block 118 backs up the Y program counter*** to the next backup Y instruction that is an X instruction boundary, and asynchronous event processing is again enabled by the block 108 (FIG. 5A) without breaking X instruction granularity.

So, when an asynchronous exception occurs, *Robinson* allows execution of all translated target instructions to complete (although *Robinson* does not describe how this is implemented), but only if no instructions in the instruction stream can generate an exception and only if the exception occurs after a particular point in the Y code sequence. For any occurrence before then, execution must be *rolled back* to a previous "granule start boundary" (Y_0, which is also the start of the source instruction X_0). Of

course, even *Robinson's* one disclosed instance of sequence execution completion is contingent on the Y code being ordered with all instructions that "update X state ... and are free of possible exceptions" last. The disadvantages of instruction roll-back, as in *Robinson's* system, are discussed not only in applicant's specification, but have also been explained at length in previous amendments.

The applicant's invention is not so restricted. Rather, as is now recited in all three independent claims 1, 15, and 16, the presence of an exception (synchronous as well as asynchronous) is sensed at any point in the translated target instruction sequence and if the sensed exception is asynchronous, execution of the target instruction sequence is resumed to completion in binary translation from the point in the translated target instruction sequence at which the asynchronous exception was sensed. Thus, the invention avoids roll-back regardless of the point in the translated instruction stream the asynchronous exception occurs. Even without *Robinson's* restrictive special ordering of the translated instructions, the invention therefore guarantees forward execution progress.

As is stated in the specification on page 20, lines 27-30 (emphasis added), "the binary translator according to the invention ... needs to efficiently handle both types of exceptions (synchronous and asynchronous), and to do so in a precise manner, that is, ***without further delay***." The guaranteed performance advantage provided by the applicant's invention as claimed is illustrated by the example on page 33 and in the discussion "Synchronization on asynchronous exceptions" beginning on page 34, line 26, which also illustrates how the asynchronous exception can occur at any point in the translated sequence. Moreover, with respect to translation and exception-handling, note the example beginning on page 37, line 13, note page 38, lines 25-26 (emphasis added): "In this example, an asynchronous exception that occurs ***anywhere*** within this sequence and generates a monitor action is handled by replacing (t17) with an INT instruction."

The amended independent claims therefore define the invention to include a feature that is lacking in *Robinson* and that provides a clear performance advantage, namely, guaranteed forward execution progress of the binary-translated target instruction sequence upon occurrence of an asynchronous exception, regardless of where in the translated target instruction sequence it arrives. Consequently, neither *Yates* nor *Robinson*, and thus no hypothetical combination of these two systems, has the novel features of the applicant's invention as claimed. The independent claims 1, 15, and 16 should therefore now be allowable.

The Examiner also rejected all the dependent claims in view of either *Yates* alone, or the same hypothetical combination of both *Yates* and *Robinson*. All of these dependent claims inherit the limitations of their respective base claim, that is, claim 1, 15 or 16. Since the independent base claims should now be allowable, so should the dependent claims.

Conclusion

The applicant's invention as defined in the independent claims includes at least one feature that is not found or suggested in the cited prior art, and that provides a substantial technical advantage. As such the applicant respectfully submits that the independent claims should be allowed, as well as the remaining, dependent claims that simply further narrow the definition of the invention.

Date: 16 June 2004

Respectfully submitted,

Jeffrey Pearce

34825 Sultan-Startup Rd.
Sultan, WA 98294
Phone & fax: (360) 793-6687

Jeffrey Pearce
Reg. No. 34,729
Attorney for the Applicant